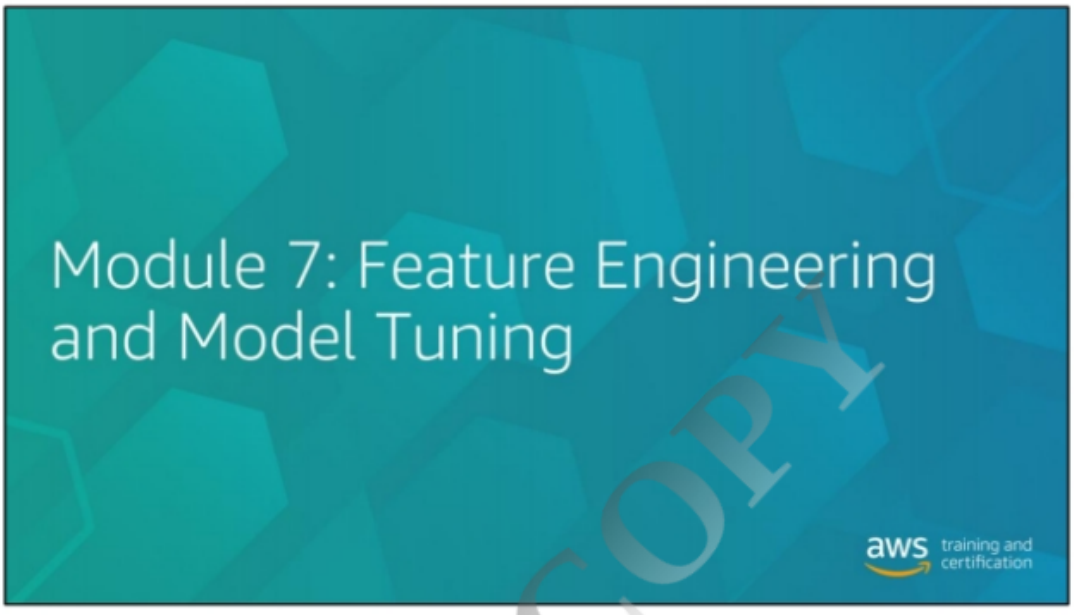
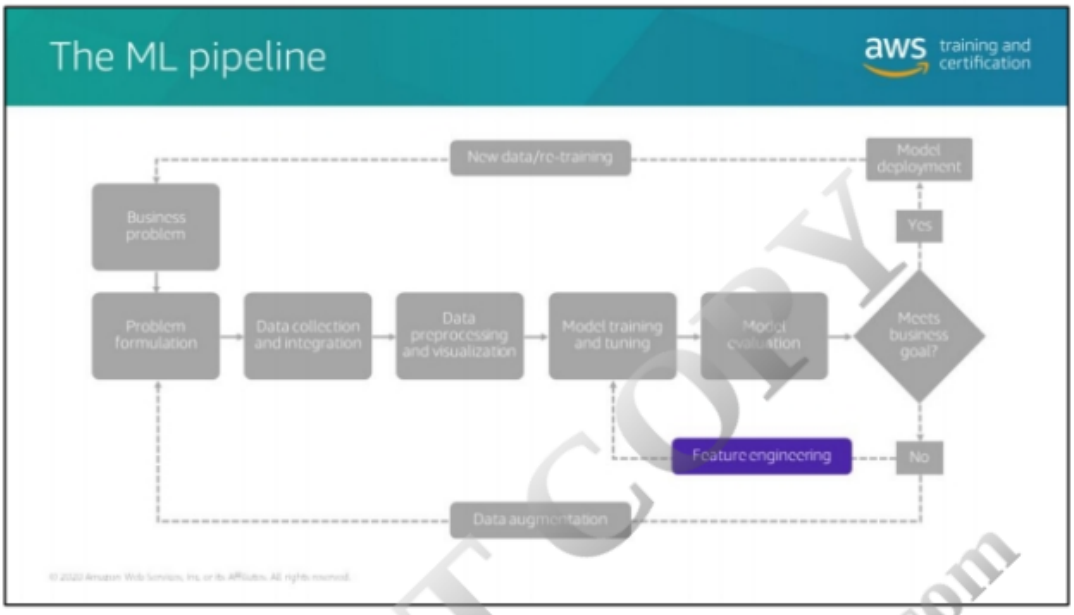
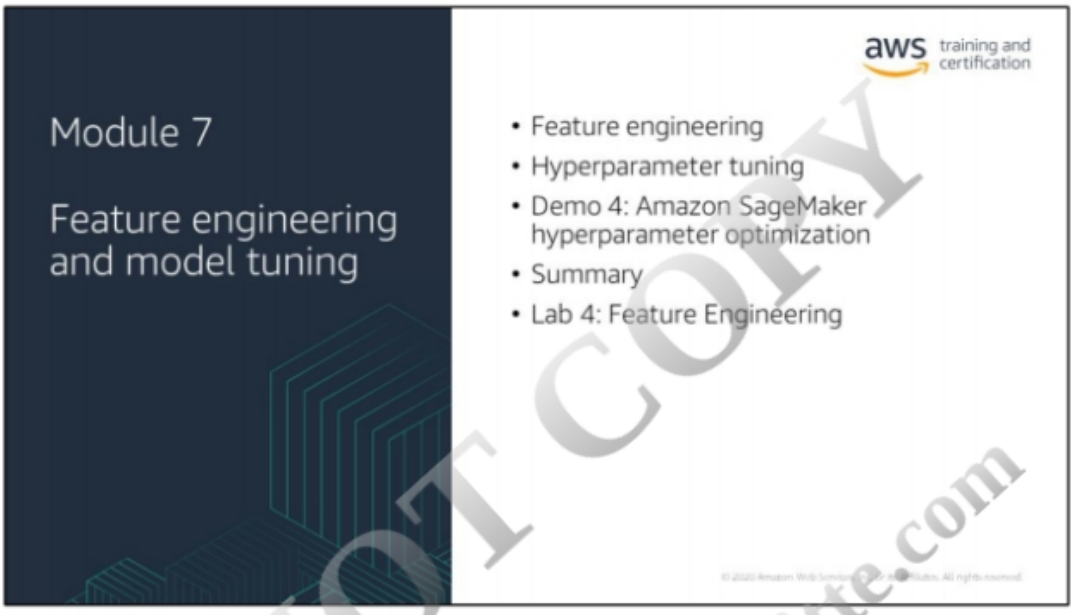
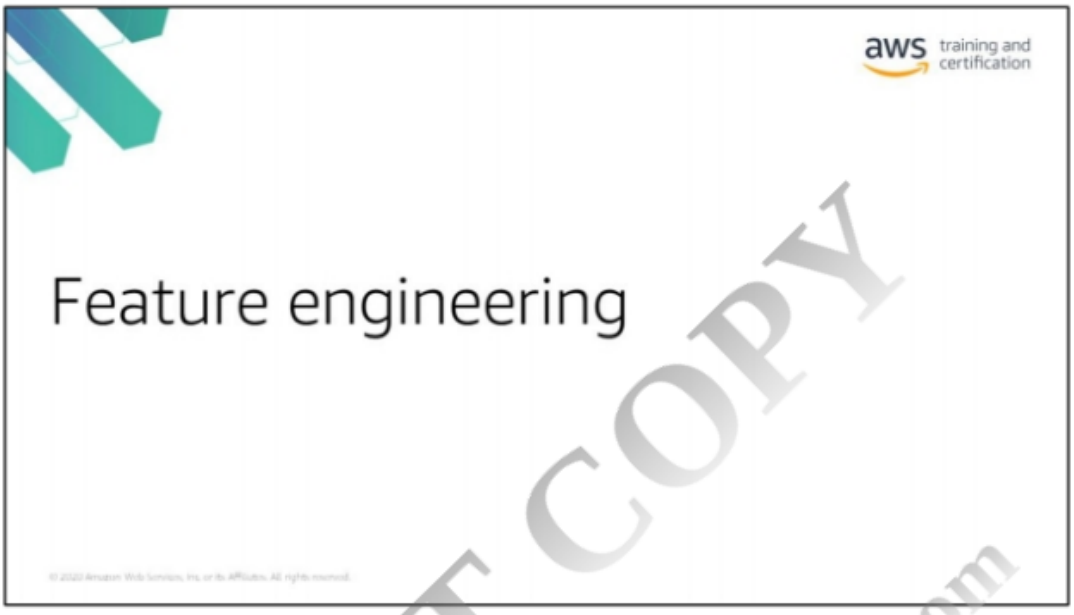


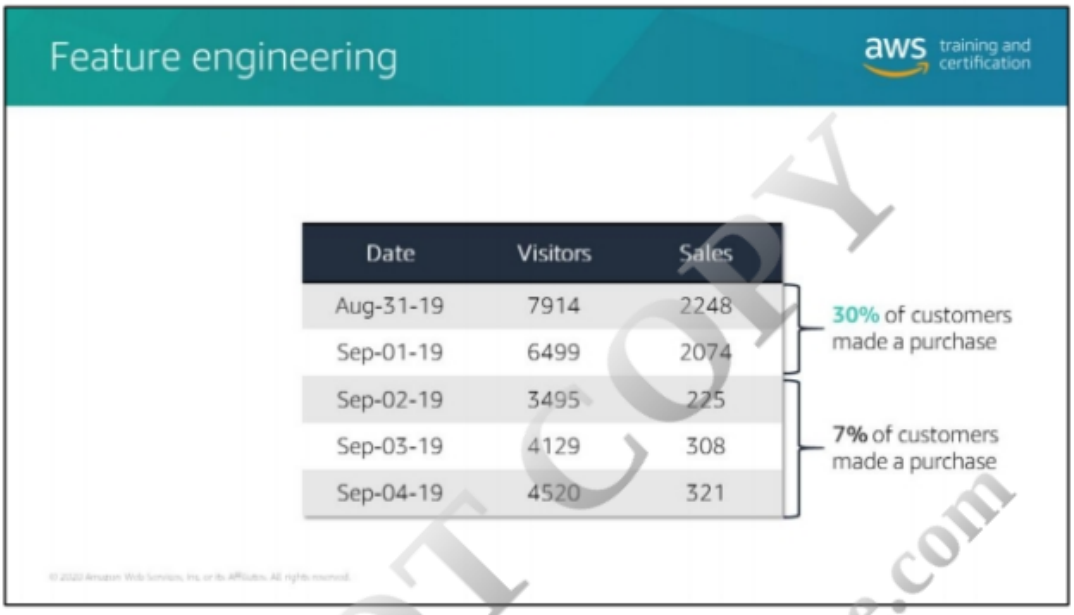


Now that you've evaluated the performance of your model for the first time, you have the information necessary to work on engineering your features and tuning your model.

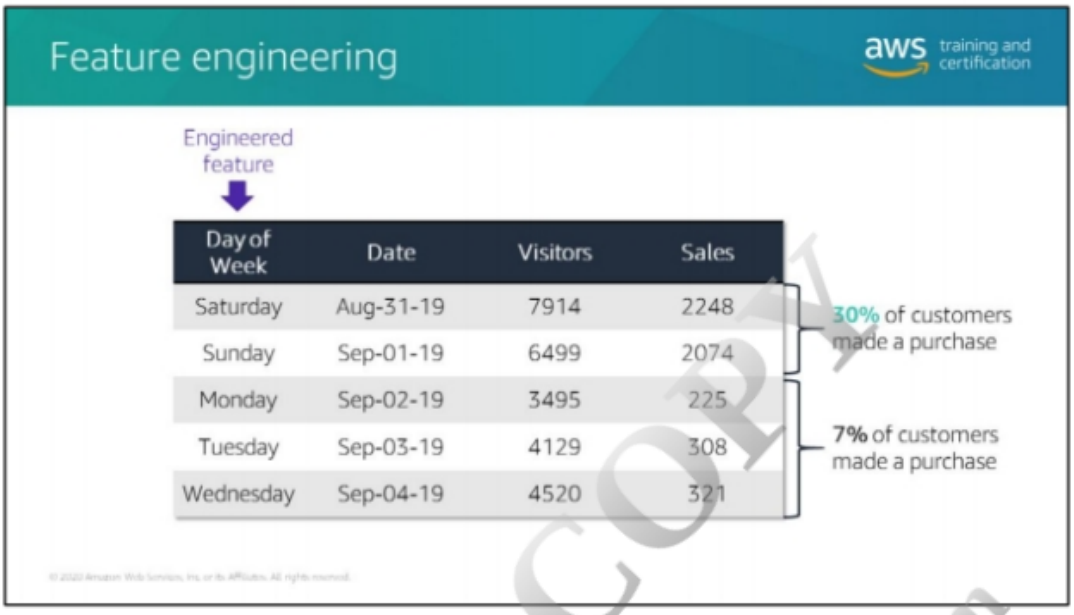


The focus of this module is on how to improve the results of your model by feature engineering and hyperparameter tuning.

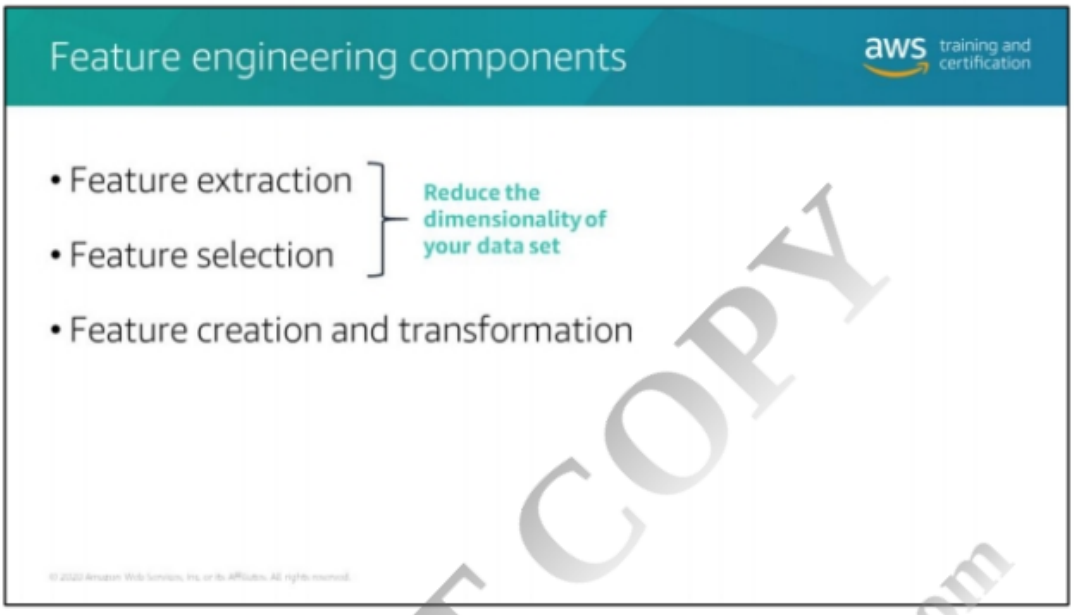




Feature engineering is the science (and art) of extracting more information from existing data in order to improve your model's prediction power and help your model learn faster. You are not adding any new data during featuring engineering, but, rather, you are making the data you already have more useful. This process often relies on domain knowledge of the data to engineer more effective features. For example, you've identified that some dates have much higher sales rates than others. What could be causing this?



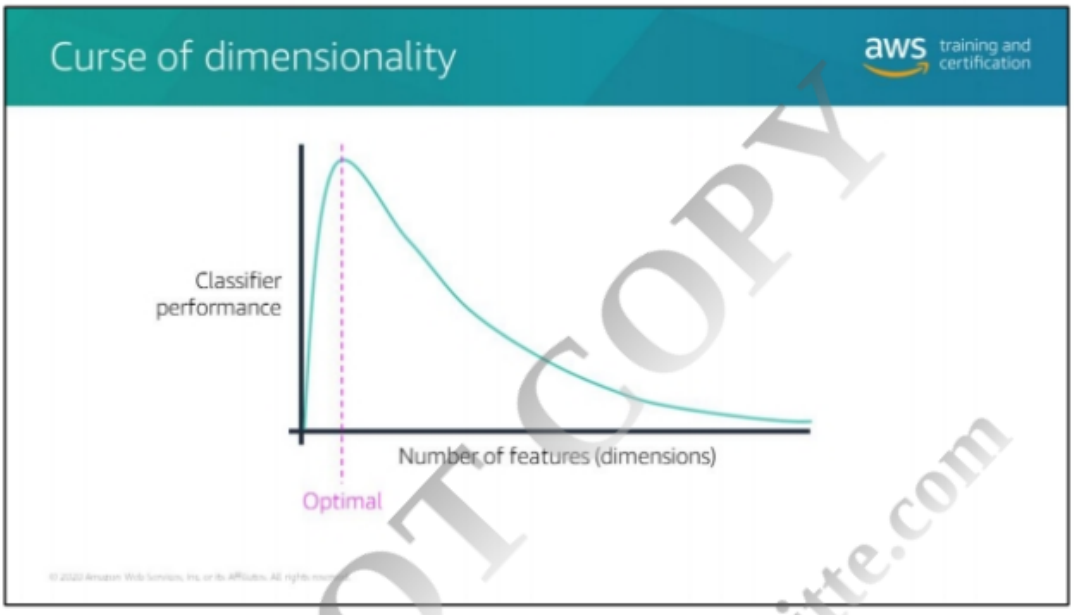
In this example, it's actually the day of the week that is influencing purchasing habits: customers are more likely to buy on the weekend. So we can engineer a feature that calls out the day of the week, then write a simple script that imputes that data automatically.



Think of feature engineering as being made up of three similar, yet slightly different, processes:

- Feature extraction
- Feature selection
- Feature creation and transformation

The first two, feature extraction and feature selection, deal with reducing the dimensionality of your data set.



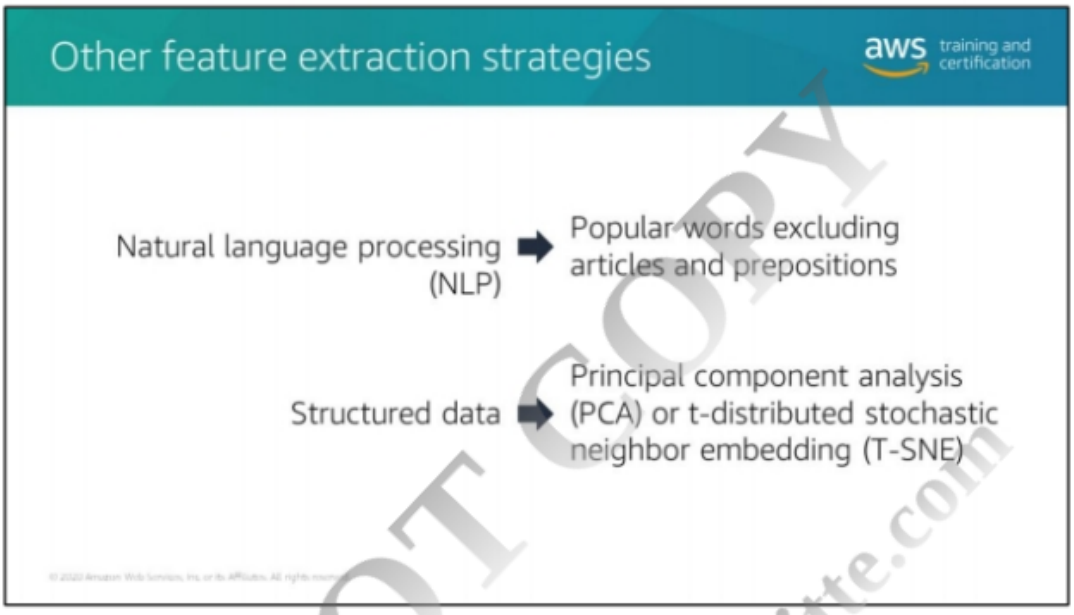
Dimensionality means the number of features (or inputs) you have in your data set. The phrase *curse of dimensionality* refers to the fact that models will have a difficult time finding the patterns you want them to identify when there are many different dimensions of data (many features) to sort through.

This is why performing feature extraction and selection is important. Let's look at both in turn.

Feature Extraction: Feature extraction is the process of automatically reducing the dimensionality of your data set by creating new features from existing features. Feature extraction can be a common process with data sets that have huge numbers of features, most often seen when working with image, audio, or text data.

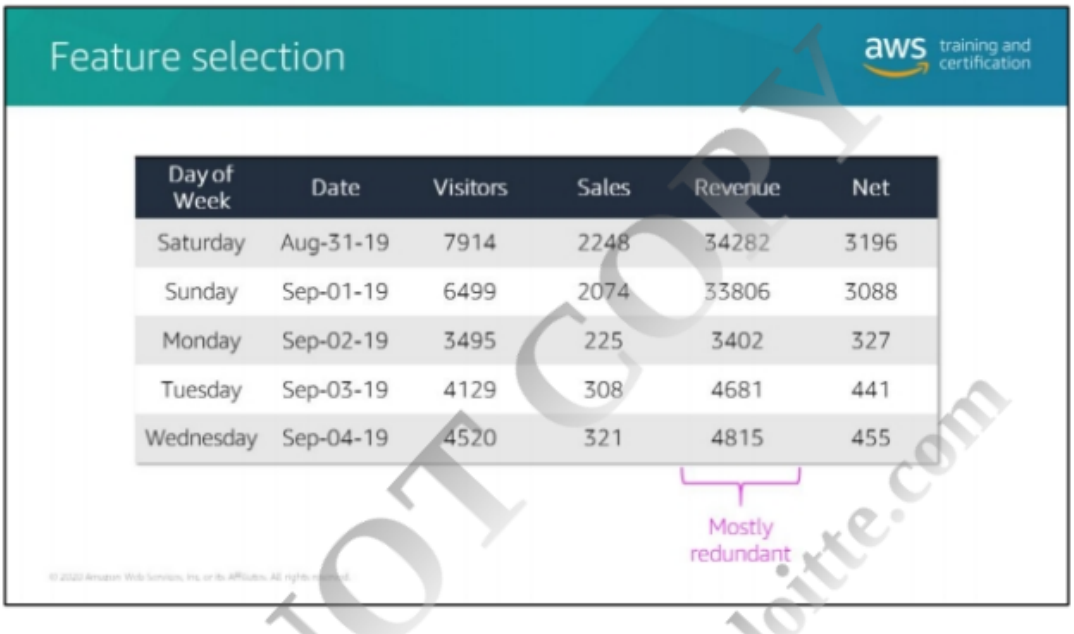
Let's look at an example of image recognition. Before the advent of neural networks, one of the ways to handle image datasets was to extract features from the individual images. If the image is a car, you extract some of the useful aspects of the image like the windshield, headlight, and tires as independent features. So instead of having raw pixels making up your dataset, you have features or columns such as windshield_*present* or headlight_*present* in your dataset. These features would make it easier for the machine learning algorithm to learn from the image data and eventually start to recognize faces.

Image source: 123RF



In most cases, you will find that the data itself will govern what specific feature extraction technique you use. For image data, it might be extracting key features like we saw above. In natural language processing, it could be extracting useful features like the most popular words from text that aren't articles or prepositions.

With structured data, you can use techniques like Principal Component Analysis (PCA) or t-distributed stochastic neighbor embedding (T-SNE) to reduce the dimensionality to a specified column length or use unsupervised approaches like clustering to cluster the data as a feature. One of the things that you should be careful of when you use feature extraction is that when you put this model into production or automate the pipeline, these features can still be replicated easily but still reduce the high dimensions in the data.



Feature Selection. Feature selection is the other common way of reducing the dimensionality of your data set. It's often used in conjunction with Feature Extraction. In other words, they're not mutually exclusive; they are often used one after the other. Feature selection ranks the existing attributes according to their predictive significance and selects the ones that are most relevant.

Not all features are created equal. There will be some features that will be more important than others to the model accuracy. There will also be features that will be redundant in the context of other features.

In this example, revenue shifts in parallel to sales, so is not likely to provide much more insight to the model than it's already getting from the sales data.

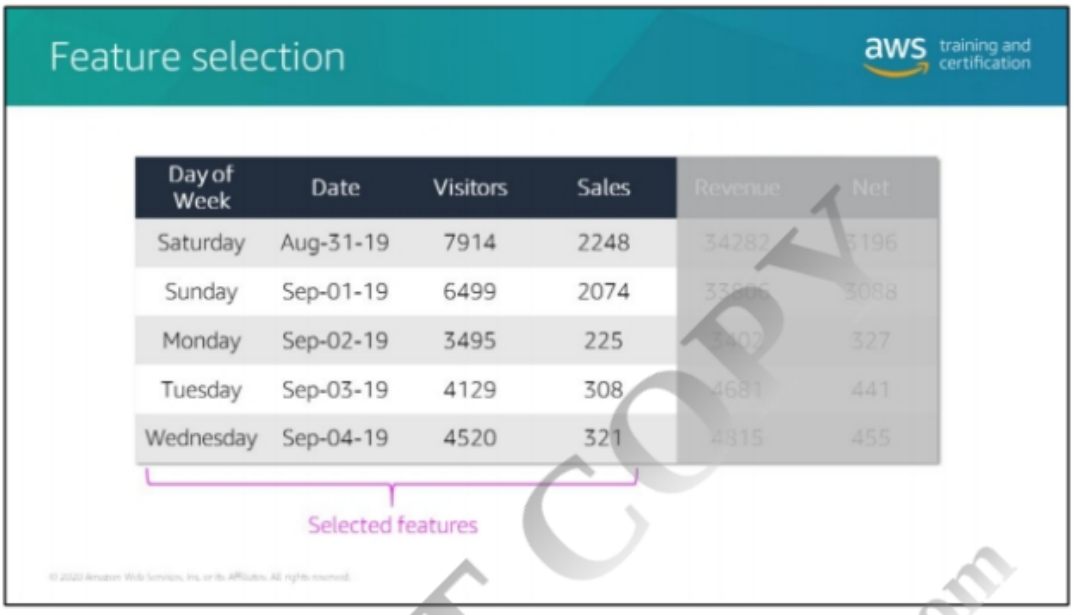
Feature selection

aws training and certification

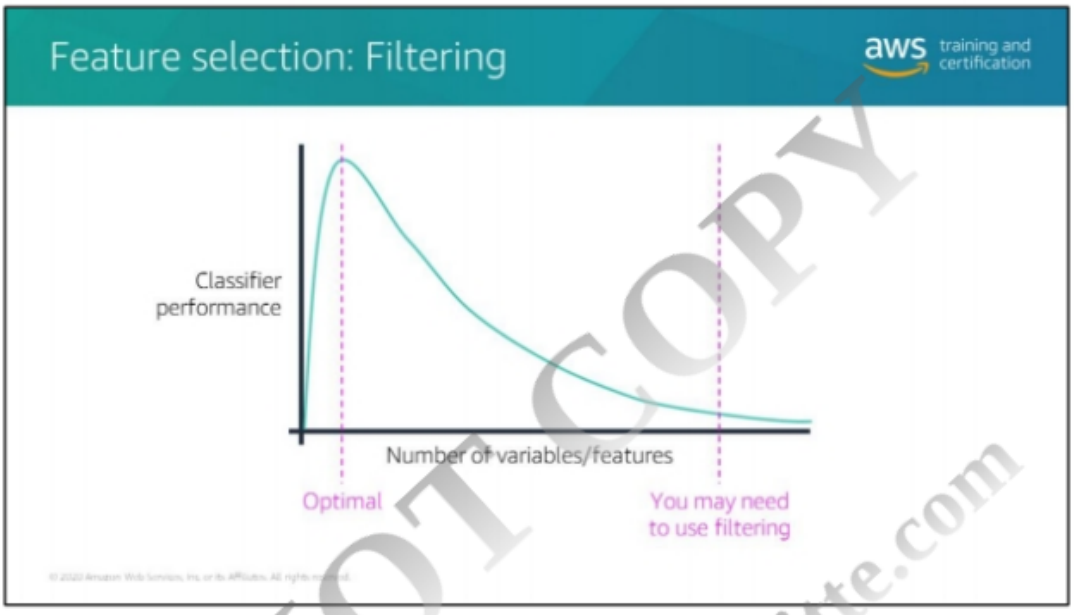
Day of Week	Date	Visitors	Sales	Revenue	Net
Saturday	Aug-31-19	7914	2248	34282	3196
Sunday	Sep-01-19	6499	2074	33806	3088
Monday	Sep-02-19	3495	225	3402	327
Tuesday	Sep-03-19	4129	308	4681	441
Wednesday	Sep-04-19	4520	321	4815	455

Not relevant

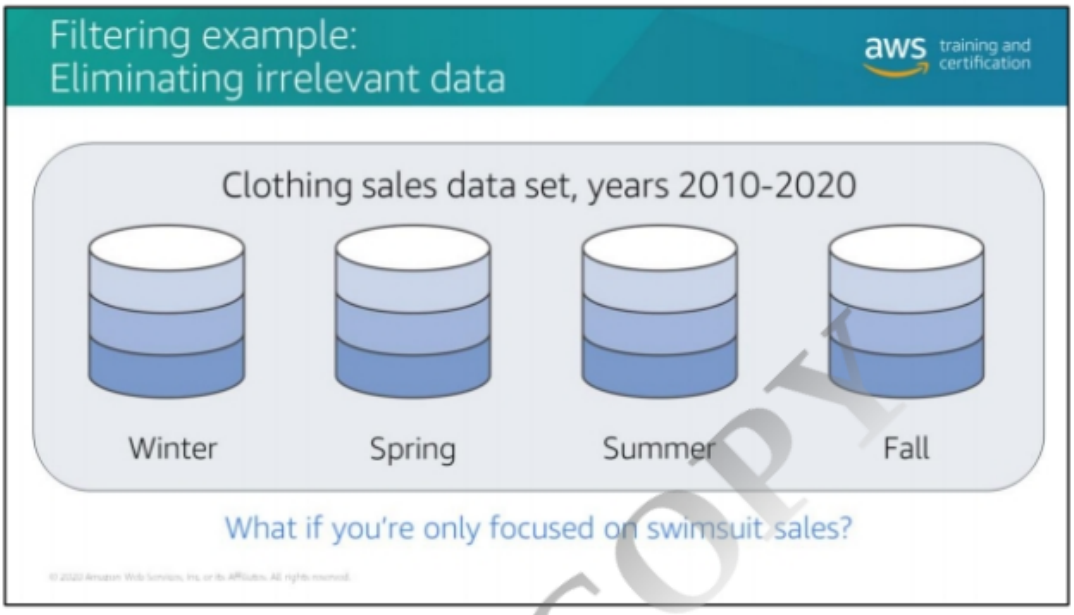
Those attributes that are irrelevant to the problem need to be removed. Feature selection addresses these problems by selecting a subset of features that are most useful to the problem. Feature selection algorithms may use a scoring method to rank and choose features, such as correlation or other feature importance methods. In this case, the business problem is about predicting sales numbers, and so Net profits from sales are probably not relevant to that prediction, so for now we can try removing that feature.



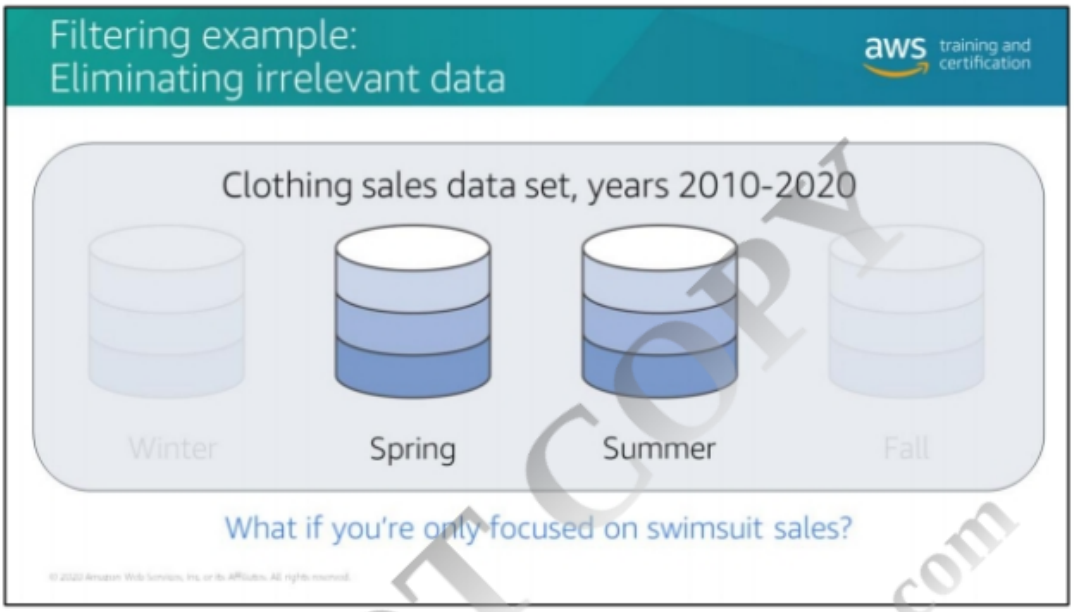
Feature selection addresses these problems by selecting a subset of features that are most useful to the problem. Feature selection algorithms may use a scoring method to rank and choose features, such as correlation or other feature importance methods.



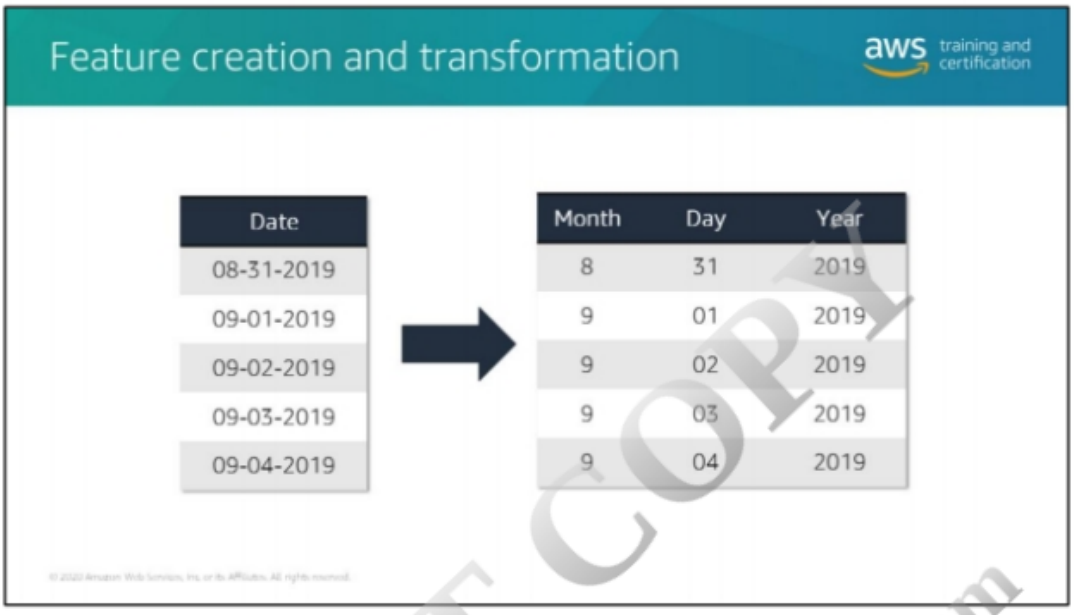
When you have too many features in your data, you should consider using filtering to reduce them. Filtering the data is one of the common techniques that you will use for feature selection. Remember, machine learning algorithms are not only used for typical structured dataset. Often times, we're dealing with images or audio, for instance. For those types of data formats, the data structure is more complicated and therefore often requires filtering to be more specific to our business problem.



As a simple example, you can take this data set for clothing sales. It includes 10 years of sales data, but your model is only trying to predict swimsuit sales? You probably don't need each year's entire sales history.

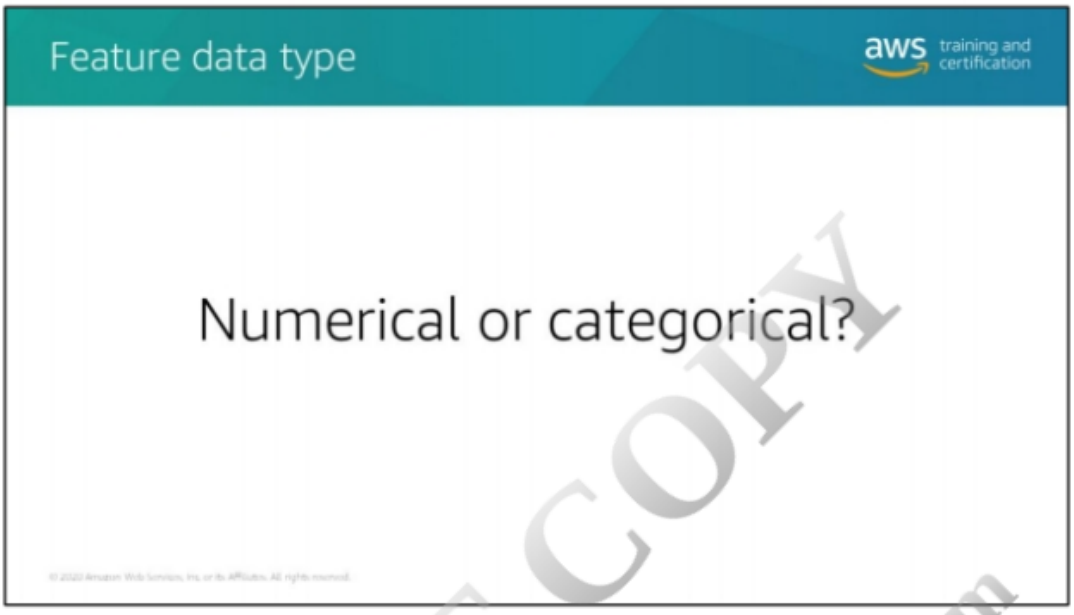


So you can filter out the months when people are far less likely to be buying swimsuits, such as in Fall and Wintertime.

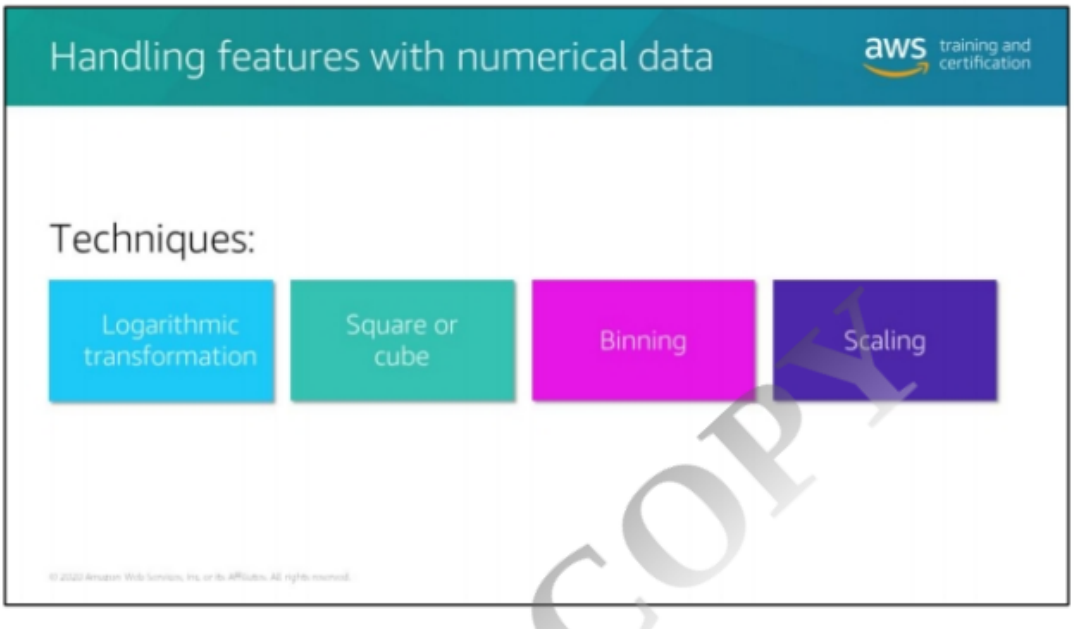


Feature Creation and Transformation: Unlike feature extraction and selection, feature creation and transformation is not a dimensionality reduction technique. Instead, feature creation and transformation is the process of generating new features from existing features. For example, say we have "date" as a feature and it's formatted as a two-digit day, two-digit month, and two-digit year (dd-mm-yy).

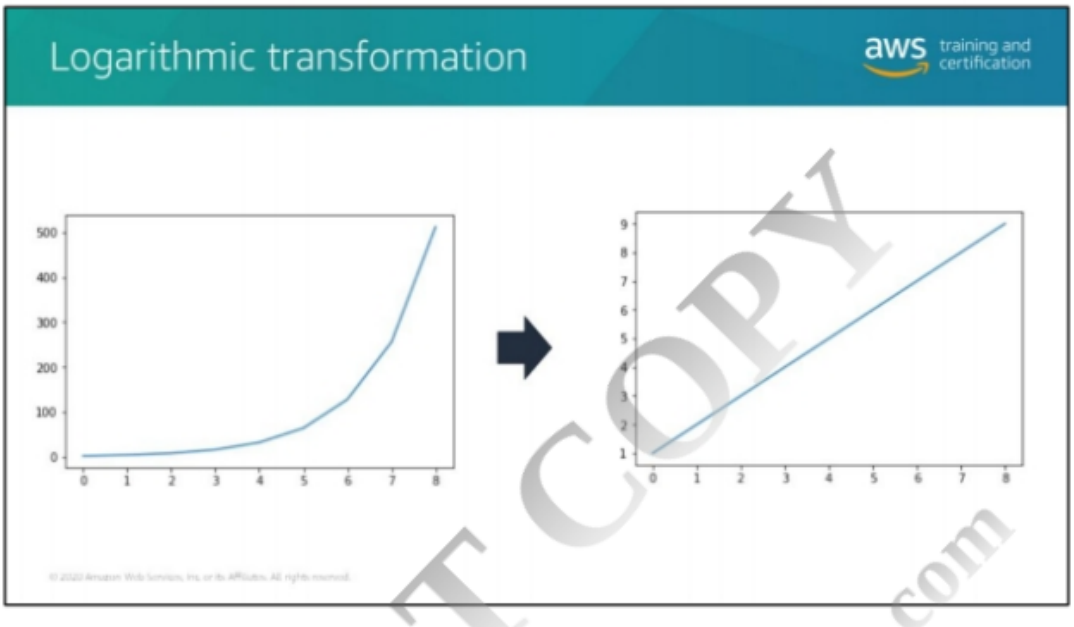
You might find the fact that combining the day, month, and year into one feature is not very helpful for your predictions. Instead, you could generate three different features, one for day, one for month and one for year, and this way potentially discover a helpful relationship between one of these features and the target.



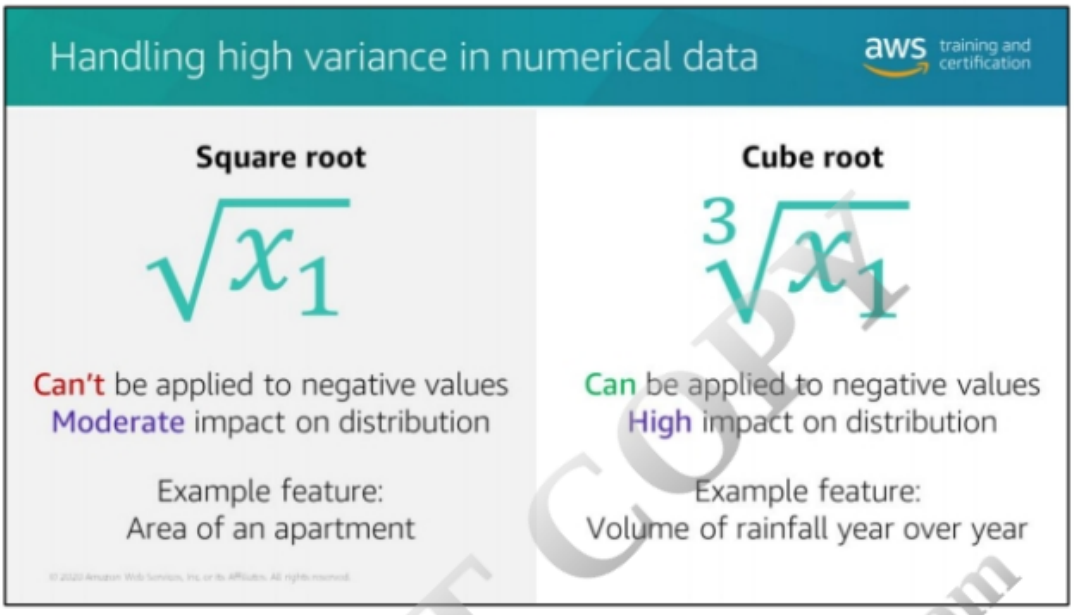
Depending on whether you're dealing with numerical or categorical data, the techniques you use for feature creation and transformation will differ. Let's look at numerical features first.



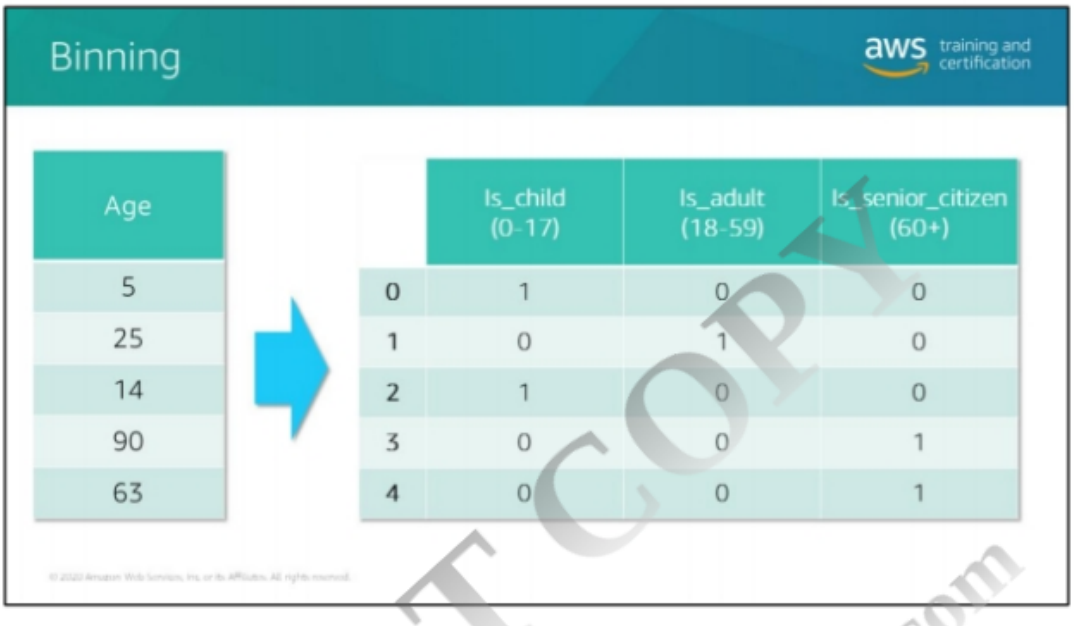
For numerical features, some of the techniques include taking the log, square root or cube root of the feature. It might also entail techniques known as binning and scaling. Let's look at some of these methods in greater detail.



Taking the log of a feature is a common transformation method used to change the shape of distribution of the variable on a distribution plot. It is generally used for reducing right skewness of features. However, it can't be applied to zero or negative values.



The square and cube root of a feature has an effect on the feature distribution. However, that impact is not as significant as logarithmic transformation. Cube root has its own advantage. It can be applied to negative values including zero. Square root can be applied to only positive values and zero.



Binning is a great strategy that puts continuous data into groups, also called bins. This is one way to convert a continuous variable into a categorical variable. Binning has a smoothing effect on the input data and may also reduce the chances of overfitting in cases with small datasets. For example, if you're looking at age, you can convert the age column into three columns: is_child, is_adult, is_senior_citizen.

Scaling

aws training and certification

Type	Bedrooms	Area (sq. ft)	Garden Size	Price	Loan Approved
House	3	2572	Small	1372000	Yes
Apartment	2	1386	N/A	699000	No
Condo	2	1932	Large	800000	No
Condo	1	851	Medium	451000	Yes
Apartment	1	600	N/A	325000	No

With numerical variables, you may also need to scale your data to avoid having one feature with more importance than the others due to their original range. This is an important technique so let's take a few minutes to really dive into it here.

Here is a piece of a home mortgage dataset used to predict loan approvals. In this example, for our typical numerical features like price, number of bedrooms, and areas, you need scaling. Scaling is a technique that is applied to each feature.

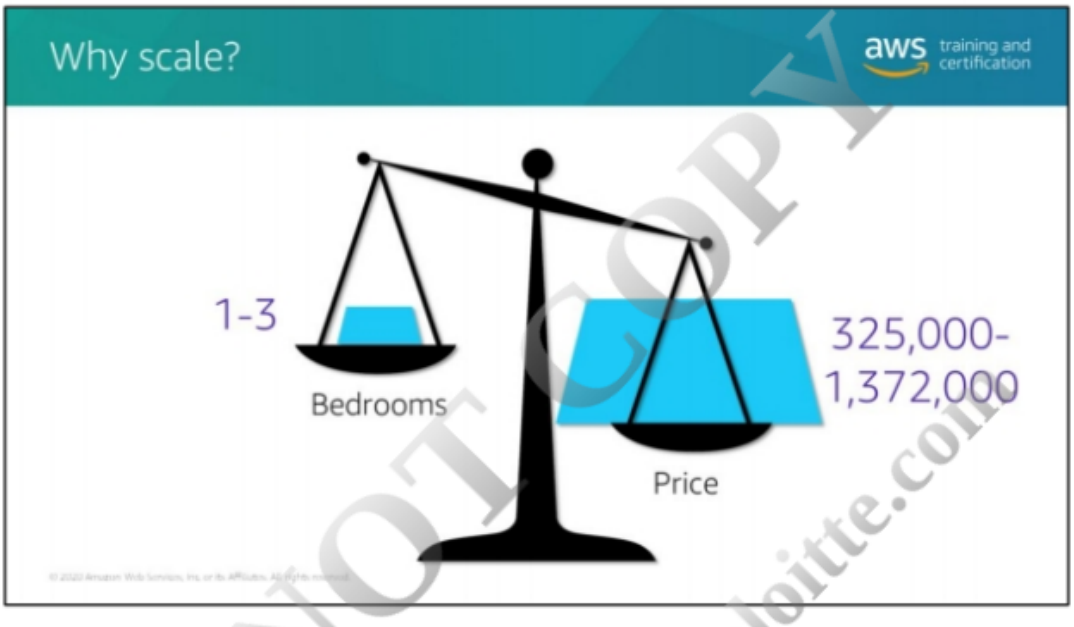
Scaling

aws training and certification

Type	Bedrooms	Area (sq. ft)	Garden Size	Price	Loan Approved
House	3	2572	Small	1372000	Yes
Apartment	2	1386	N/A	699000	No
Condo	2	1932	Large	800000	No
Condo	1	851	Medium	451000	Yes
Apartment	1	600	N/A	325000	No

For each column in this particular dataset, we want the value to be between minus one and plus one. The main reason for scaling is that the range of these features is dramatically different.

For example, here, Bedroom has a value of one, two, or three. There are not many bedrooms in the house, but on the other hand, the price of the house ranges from around $300,000 to over 1.25 million dollars. That's a huge difference in scale, and that can give disproportionate influence to larger scaled variables

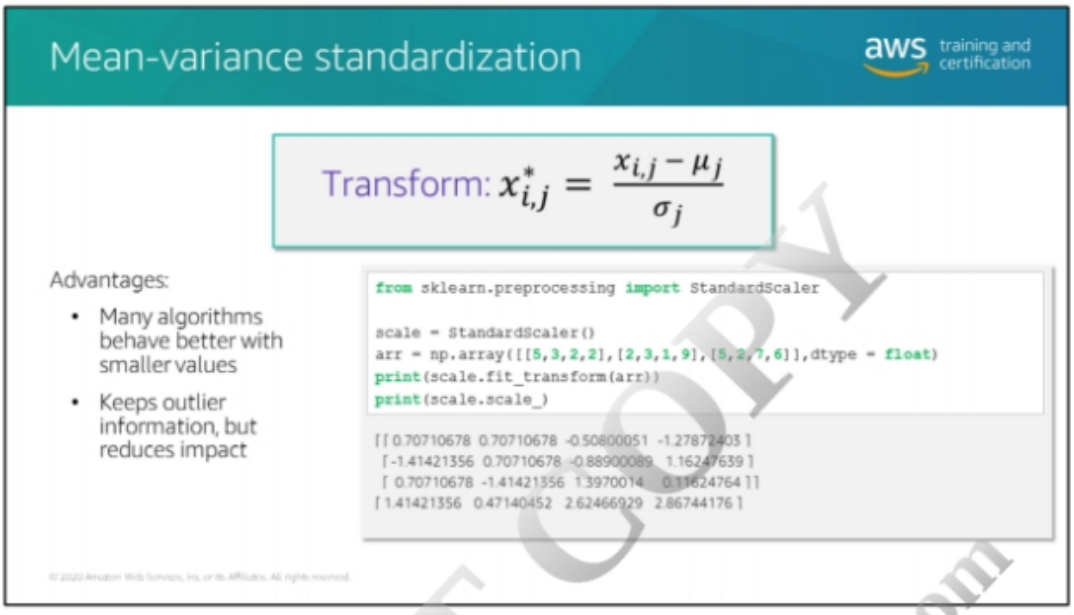


Scaling is important because a lot, although not all, machine learning algorithms are very sensitive to different ranges of data. Sometimes, a wide range of data will lead to optimization failure. There are exceptions, however. Decision trees in the random forest algorithm, are usually not sensitive to the scale of the variables in the dataset.

So, in many cases the solution is aligning all the features onto the same scale. But how do we do that?

Here is a collection of commonly used scaling techniques that are usually used for data science and machine learning projects.

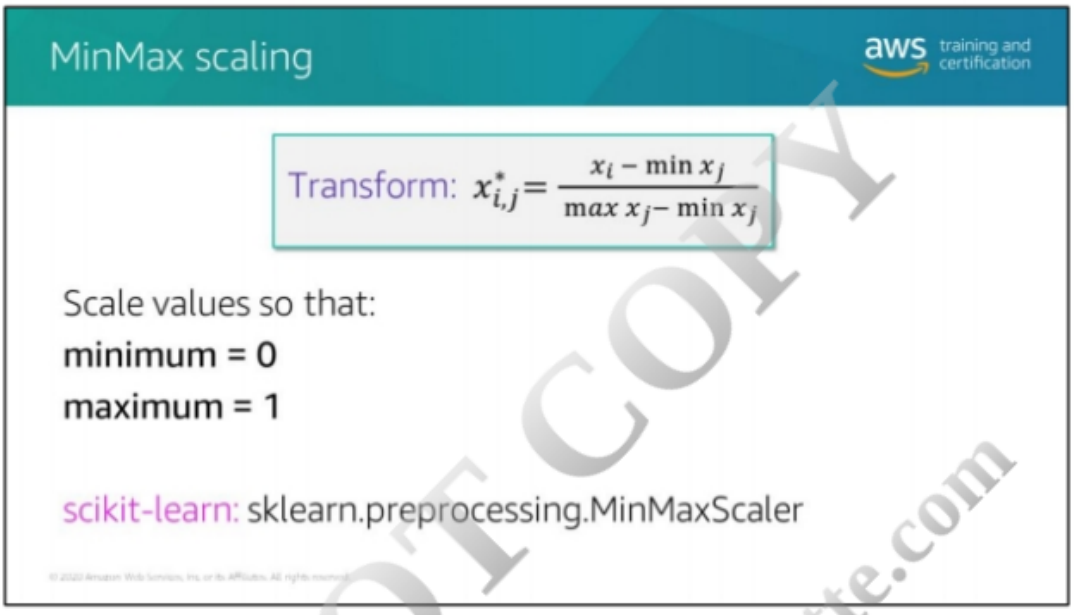
In general, we're applying a transformation to a particular column, and different columns are scaled independently. We are only using the data in a specific column to do the scaling.



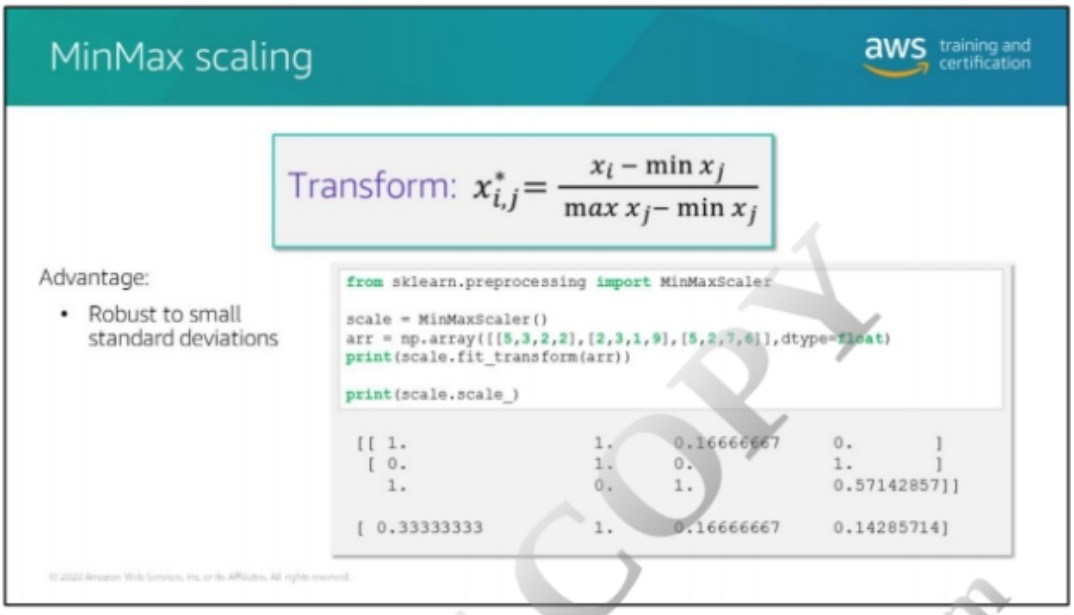
The widely used scaler is standard deviation methodology using mean and the variance. For one particular feature X, we are going to first find out the mean and standard deviation for that particular feature in the training dataset. What we're going to do is remove the mean and divide by the standard deviation for each observation. After scaling, the new feature is going to have a mean of zero, and a standard deviation of one. To achieve this transformation easily, we can call the scikit-learn StandardScaler function.

After using the mean-variance standard deviation algorithm the features will be on the same scale with mean of zero and standard deviation of one. So the optimization algorithm for many machine learning methods will be more likely to converge.

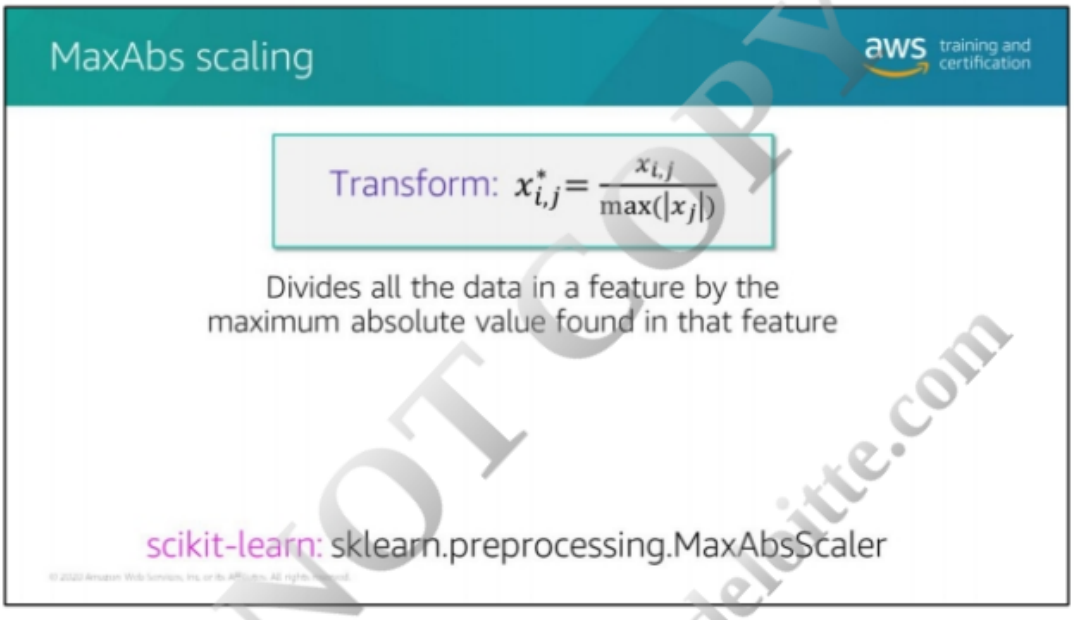
Also, by using the mean-variance standard deviation procedure, we still keep the outliers, but reduce the impact of those outliers dramatically.
Scaled values are centered around mean $\mu_j=0$, with standard deviation $\sigma_j=1$ for each data column.



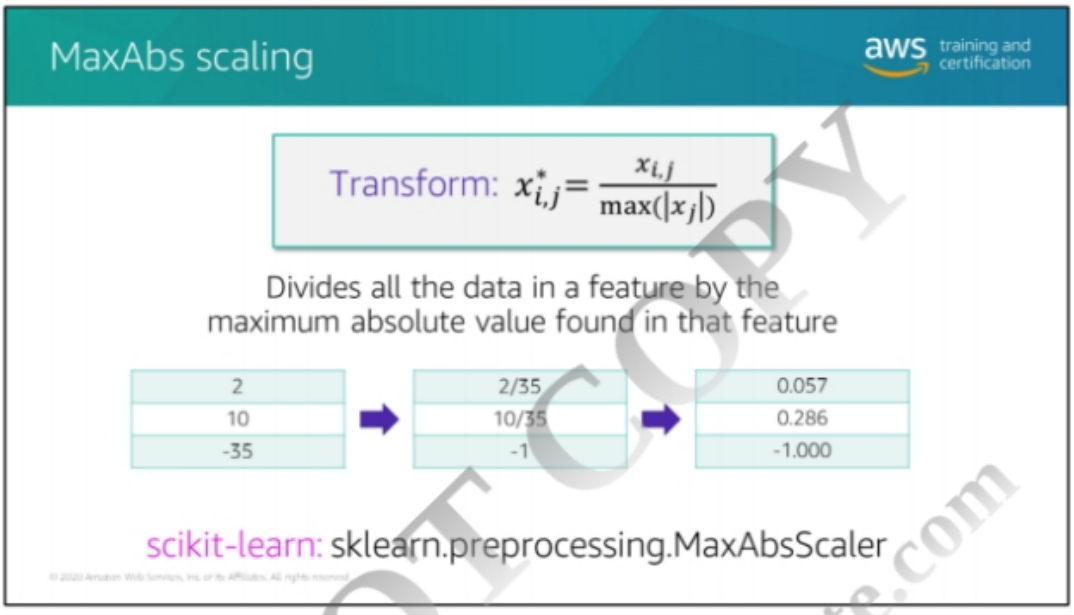
Another widely used scaling method is MinMax scaling. In this particular method, we start by finding the minimum value and the maximum value for the feature we want to scale. Then, we subtract the minimum value of the feature and divide it by the difference between the maximum of the feature and the minimum of the feature. After the transformation, the new variable is going to be between zero and one. MinMax scaling can be easily done by calling the MinMax scaler from the scikit-learn package.



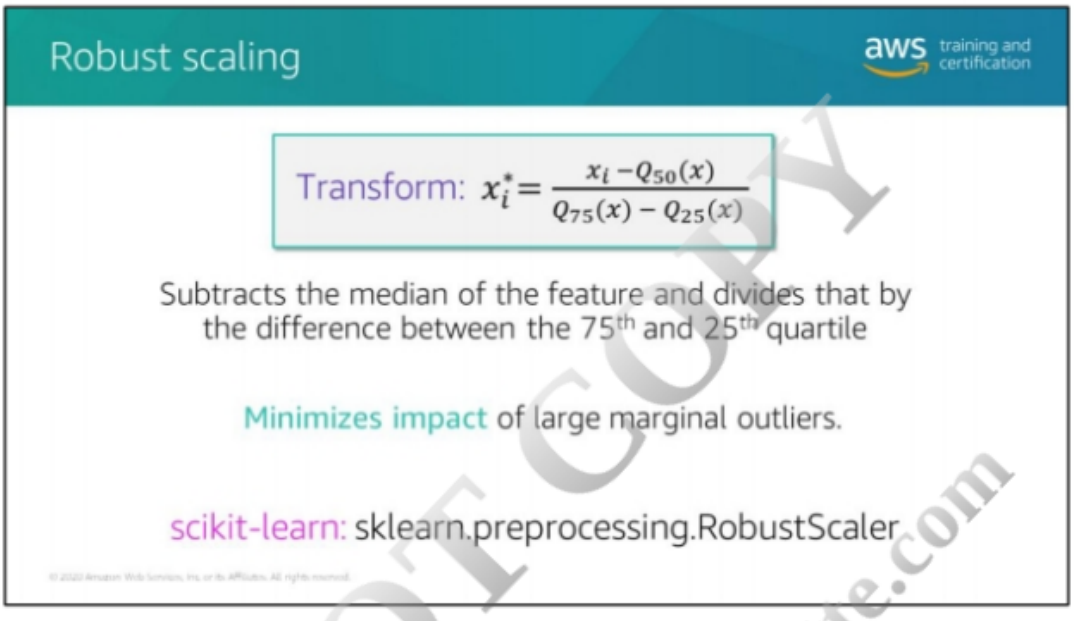
The advantage of MinMax scaling is that it is very robust for cases with small standard deviation. Remember, the difference between the maximum and minimum is usually larger than the standard deviation. When the standard deviation is small the mean-variance scaling is not going to be robust because we're dividing by a very small number. This transformation does not change the distribution of the feature and due to the decreased standard deviation the effects of the **outliers** increases. Therefore, before transformation, it is recommended to handle the outliers.



There is another scaling method called MaxAbs scaling. For the particular feature in the training data, we first take the absolute value and then find the maximum of the absolute value of that feature. Once we find that, we're going to divide every element by the maximum of the absolute value for that particular feature. MaxAbs scaling doesn't destroy sparsity because we don't center the observation through any measurement, we just scale it. The MaxAbs scaler can be called from the scikit-learn library as well.



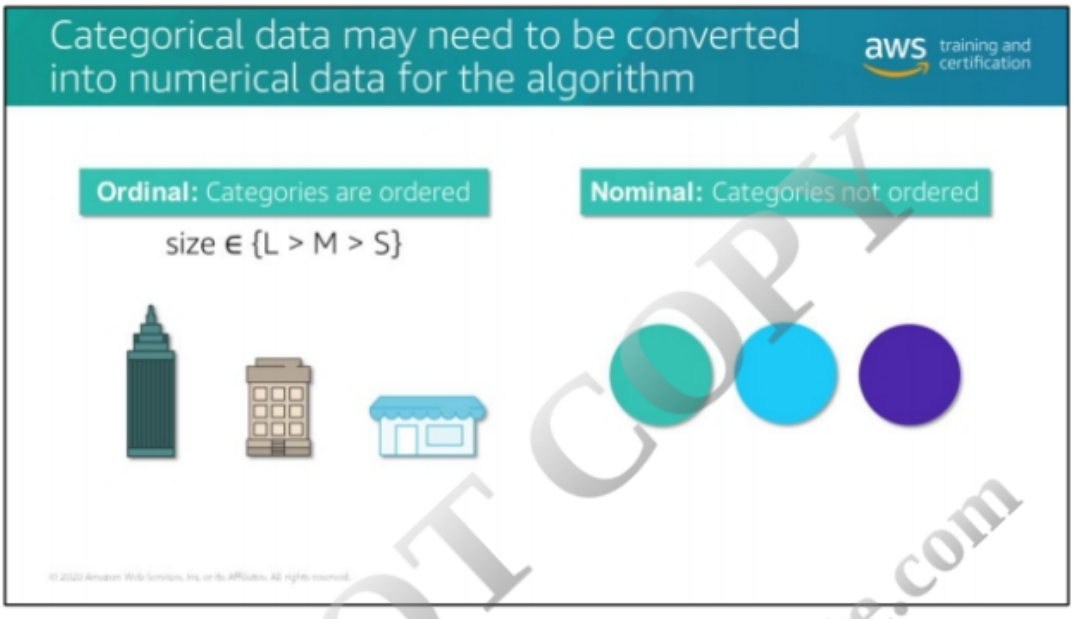
There is another scaling method called MaxAbs scaling. For the particular feature in the training data, we first take the absolute value and then find the maximum of the absolute value of that feature. Once we find that, we're going to divide every element by the maximum of the absolute value for that particular feature. MaxAbs scaling doesn't destroy sparsity because we don't center the observation through any measurement, we just scale it. The MaxAbs scaler can be called from the scikit-learn library as well.



There's another scaling method called Robust scaling. For Robust scaling, we first look at the particular feature in the training dataset and find the median, the 75th quartile, and the 25th quartile for that particular feature in the training data. Once we figure out those three quantities, we can calculate the Robust scaled variable by subtracting the median of the feature and then dividing it by the difference between the 75th and 25th quantile of that feature in the training data.

After the transformation it will be *robust* to outliers, because outliers will have minimal impact when calculating medians and quartiles. Robust scaling can be done by using the robust_scale function in scikit-learn.

For additional diagrams to help explain the concepts please review the visual diagrams available on the scikit-learn web site at https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#results



Now let's shift to transformation techniques for categorical data. When dealing with categorical data, you'll often need to convert that data into numerical data before it can be read by your ML algorithm. Your approach will differ depending on whether your data is ordinal (the categories are ordered) or nominal (categories are not ordered).

Categorical data may need to be converted into numerical data for the algorithm

aws training and certification

Type	Bedrooms	Area (sq. ft)	Garden Size	Price	Loan Approved
House	3	2572	Small	1372000	1
Apartment	2	1386	N/A	699000	0
Condo	2	1932	Large	800000	0
Condo	1	851	Medium	451000	1
Apartment	1	600	N/A	325000	0

Let's look back at the housing dataset we just introduced. The dataset includes several features: house type, number of bedrooms, area of home, garden size, and home price. There's also a yes/no value as the target variable, indicating whether a home loan was approved or not.

Categorical data may need to be converted into numerical data for the algorithm

aws training and certification

Type	Bedrooms	Area (sq. ft)	Garden Size	Price	Loan Approved
House	3	2572	5	1372000	1
Apartment	2	1386	0	699000	0
Condo	2	1932	20	800000	0
Condo	1	851	10	451000	1
Apartment	1	600	0	325000	0

House type and garden size represent categorical features. Garden size, more specifically, represents ordinal data, because Small, Medium, and Large can be represented in an order (note: N represents "no garden").

For ordinal variables like this one, you can use a map function in Pandas to convert the text into numerical values. For example, you can define the relative difference for those different categories in the ordinal variable.

Often, the numerical value you provide in the mapping is derived from your business insight of the dataset and the business itself. For the garden size S, you can use 5; for M, use 10; for L, use 20; and for N, use 0.

You can easily apply the map function from Pandas to replace the categorical variable with the numerical value. It is a one-to-one mapping.

Not all categorical data should be converted

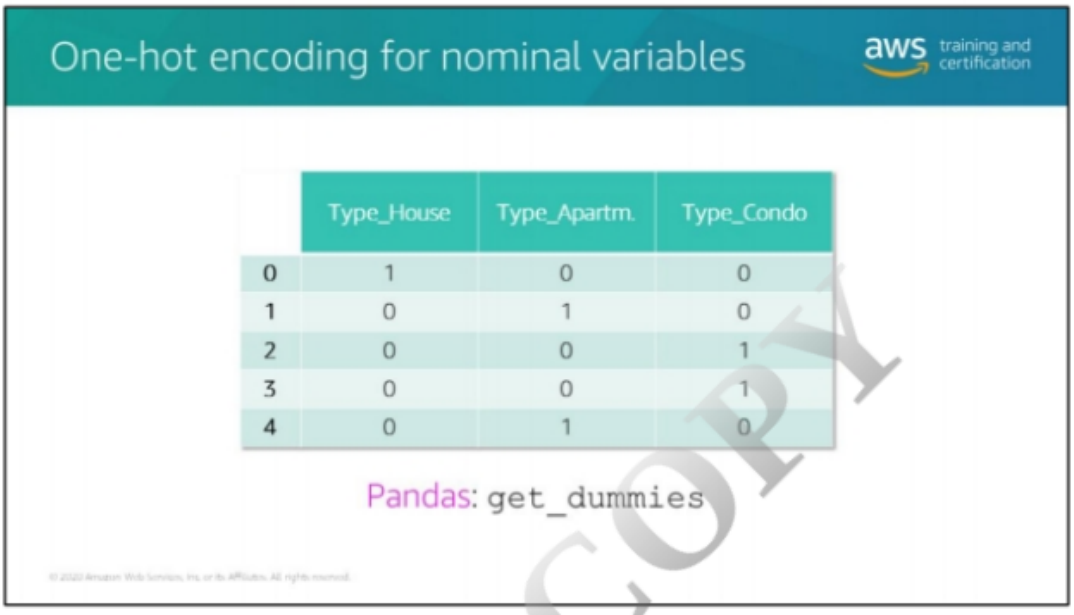
aws training and certification

Type	Bedrooms	Area (sq. ft)	Garden Size	Price	Loan Approved
House	3	2572	5	1372000	1
Apartment	2	1386	0	699000	0
Condo	2	1932	20	800000	0
Condo	1	851	10	451000	1
Apartment	1	600	0	325000	0

A numerical value would imply an order of difference between the values that may not really exist

By contrast, it's not generally recommended to encode nominal variables like home type to numerical data. If you encode this variable or feature into integers, it becomes one, two, and three.

One, two, and three really implies that something has a numerical value. They have order difference, and there is also a magnitude to the difference between the numbers. These additional features are artifacts that do not belong to the original data. And these artifacts may give you the wrong or unexpected results.

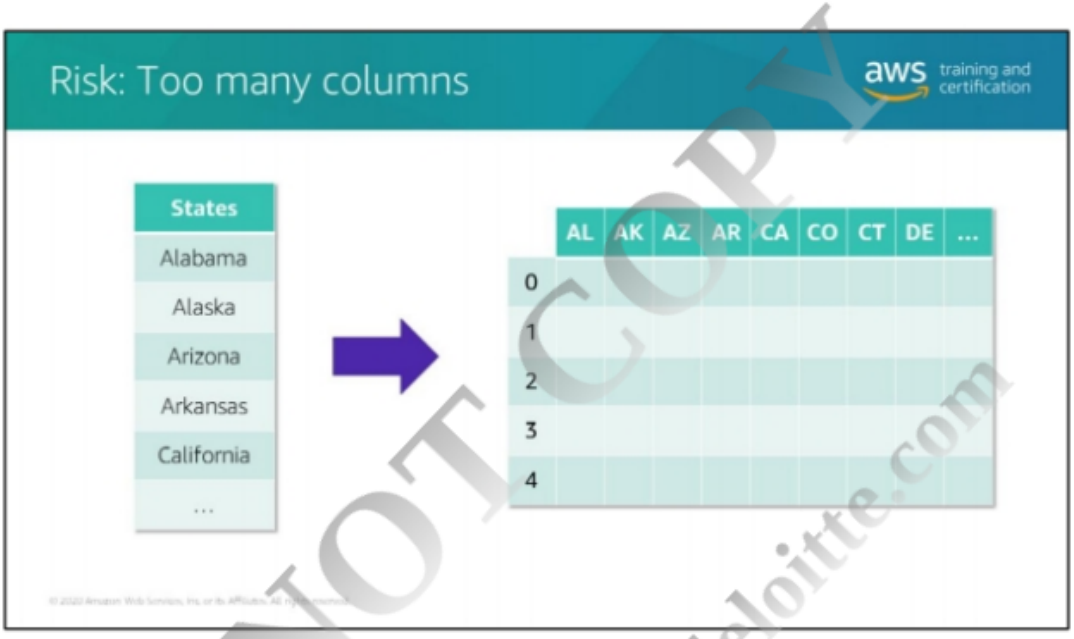


So how do you encode nominal variables? The one-hot encoding method is a good choice. Here's how it works.

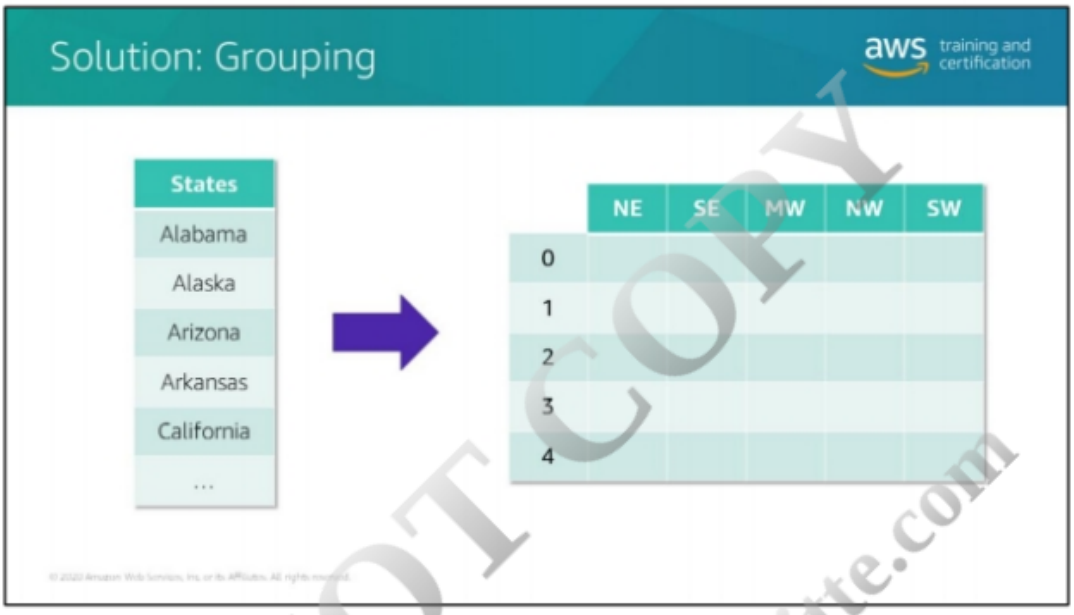
In this example, you have the one column called House Types, and three different levels: House, Apartment, and Condo. The data frame has five observations for that particular feature.

With one-hot encoding, you convert this one column of Home Types into three columns: A column for House, a column for Apartment, and a column for Condo. You encode each observation with either a 1 or 0: 1 to indicate the home type of that particular observation, or 0 for the other options.

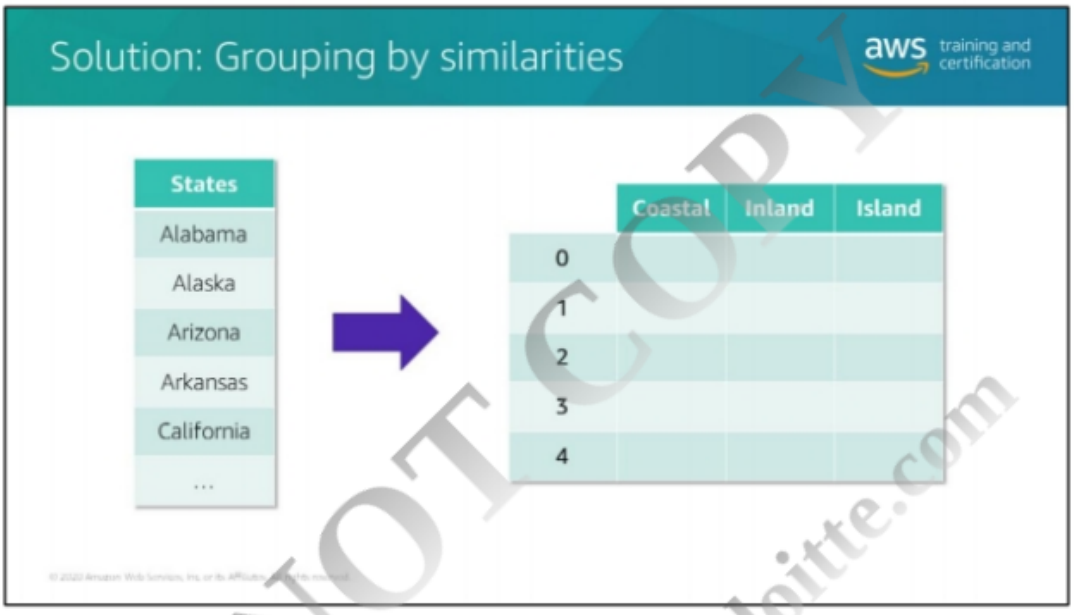
Pandas get_dummies function will automatically create the new columns based on one-hot encoding and create the column names for you, with the entry for each category at the end of the variable name. Once you have a dummy variable, or one-hot encoded variable to represent the original feature, you can use those three features to replace the one feature, Home Types.



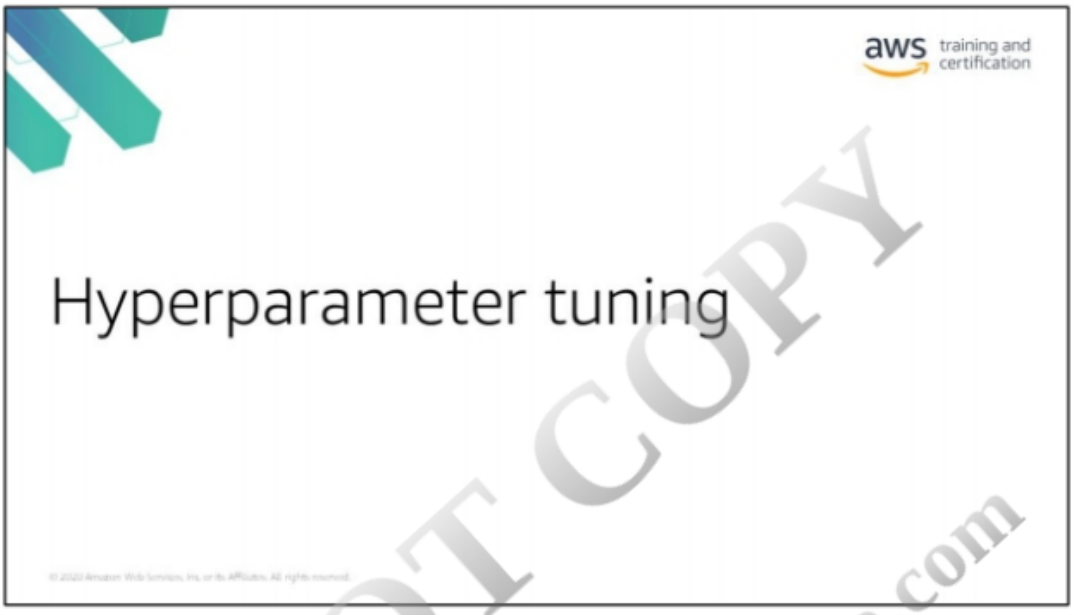
One-hot encoding can occasionally lead to the creation of many new columns, increasing your dataset dramatically and adding to a higher dimensionality or number of features. For example, we can define a hierarchical structure. Say we have a column made up of US States and we want to one-hot encode that column so each state has it's own column. In that case, we're adding close to 50 more columns to our dataset.

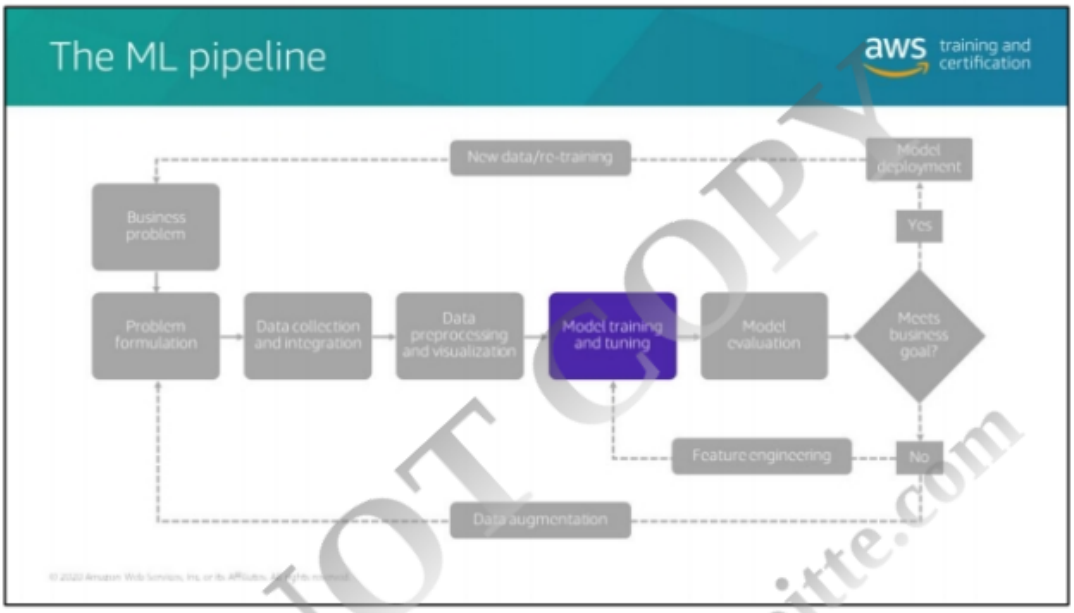


Instead, we could aggregate different states by regions to reduce the number of levels in that category. For instance here, we've grouped them into Northeast, Southeast, Midwest, Northwest, and Southwest states.

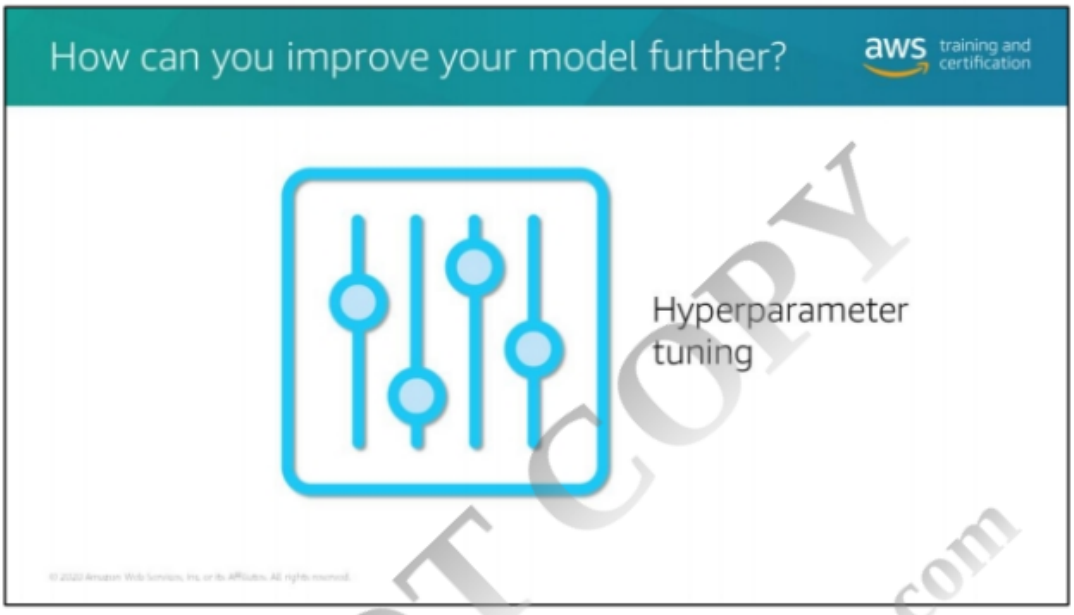


We can also try to group the levels by similarities . The similarities can be defined by different kinds of measurement. Once we group those levels by similarity to new groupings, the new groupings will have fewer levels. We can use that as the starting point.



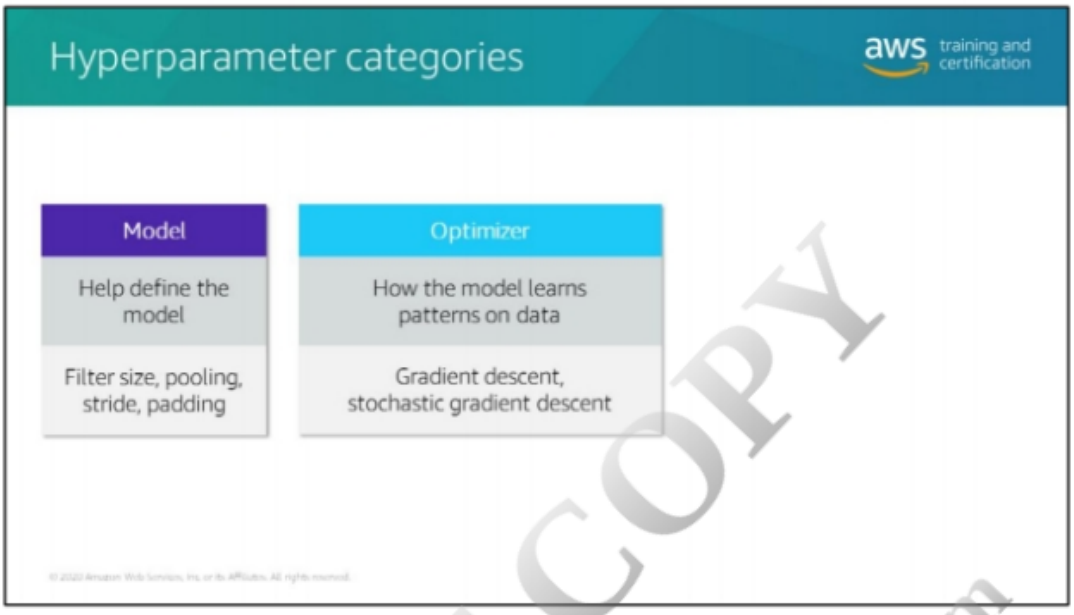


So now that you've selected, engineered, created, and grouped your features as necessary, you can move on to tuning the hyperparameters of your model to provide even stronger accuracy.

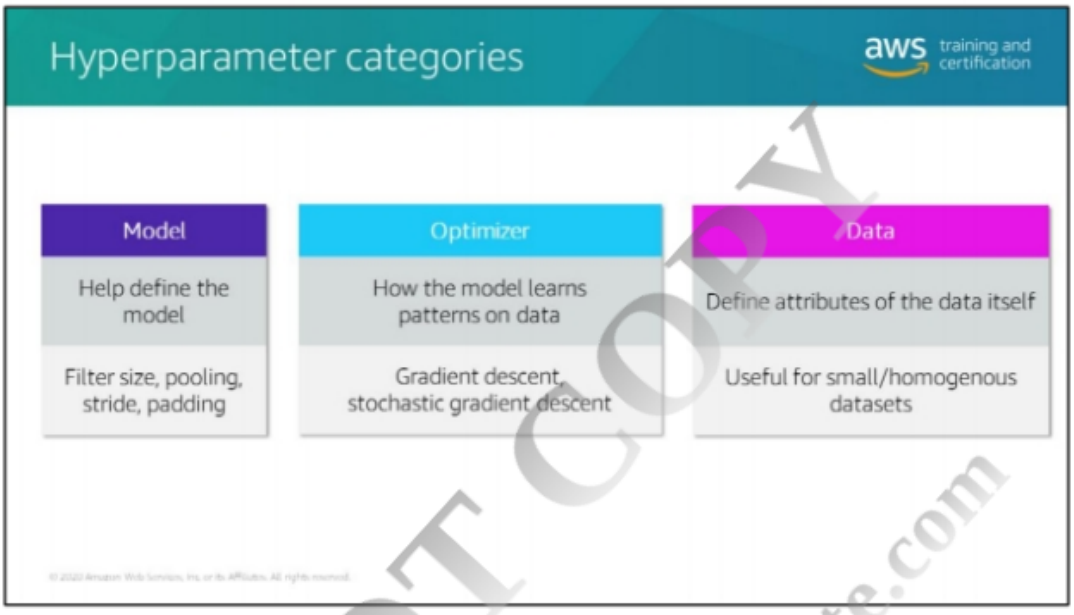


Hyperparameters can be thought of as the knobs used to tune the machine learning algorithm to improve its performance. Now that we're looking more explicitly at tuning our models, it's time to look more specifically at the types of hyperparameters out there and how to go about performing hyperparameter optimization.

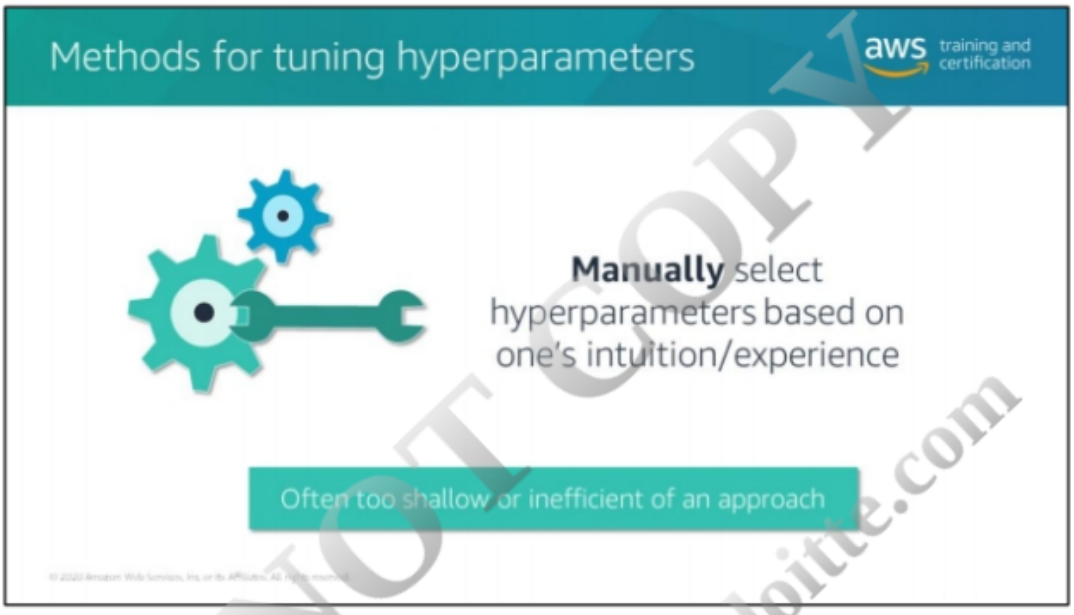
There are a couple different categories of hyperparameters. The first kind are model hyperparameters that help define the model itself. Some neural network-based models, for instance, need you to define an architecture before you can start training them. The architecture will include a specific number of layers in the neural network and the activation functions used within. In a neural network for a computer vision problem, for example, additional attributes of the architecture need to be defined, like: filter size, pooling, and the stride or padding.



The second kind are *optimizer* hyperparameters, which are related to how the model learn the patterns based on data and are used for a neural network model. These types of hyperparameters include optimizers like gradient descent and stochastic gradient descent. These can also include optimizers using momentum like Adam or initializing the parameter weights using methods like Xavier initialization, or He initialization.

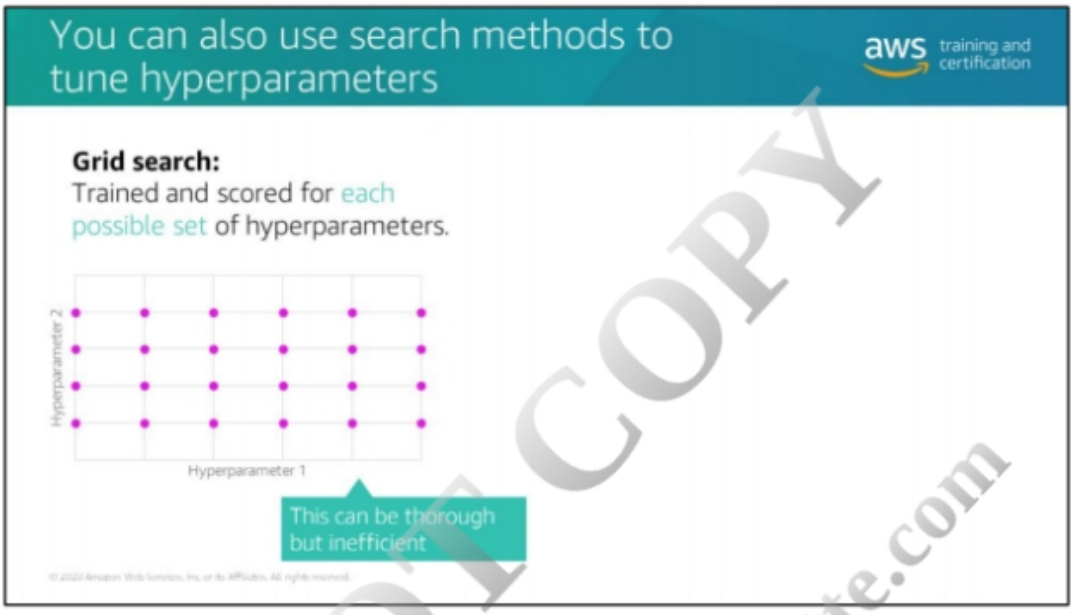


The third kind are *data* hyperparameters, which relate to the attributes of the data itself. This includes attributes that define different data augmentation techniques like cropping or resizing for image-related problems. They are often used when you don't have enough data or enough variation in your data.

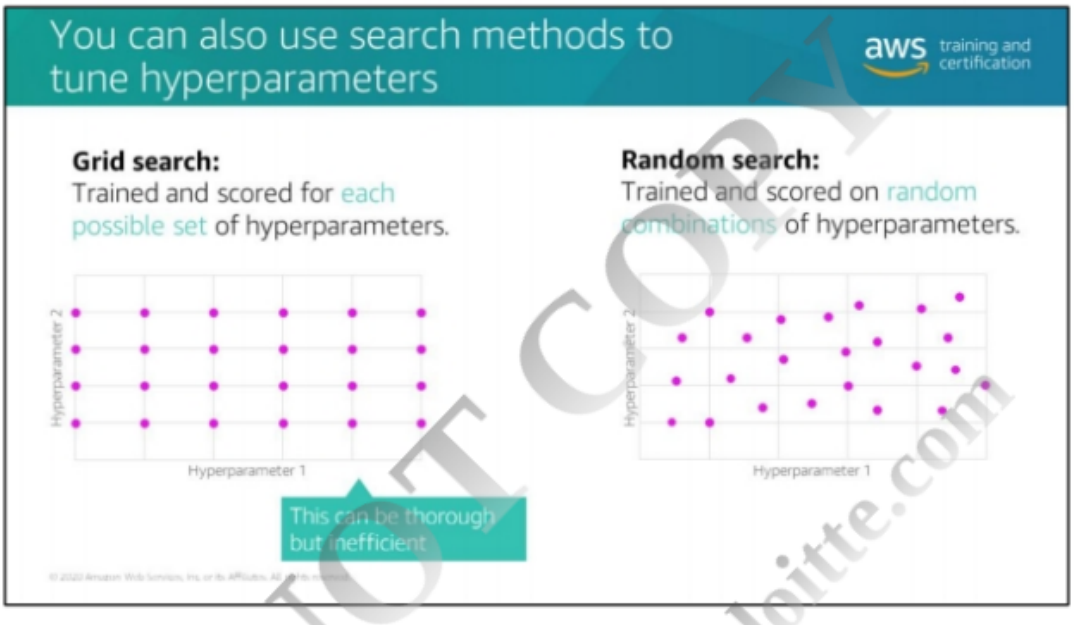


Tuning hyperparameters can be very labor-intensive. Traditionally, this was done manually: someone who has domain experience related to that hyperparameter and the use case would manually select the hyperparameters based on their intuition and experience. Then they would train the model and score it on the validation data. This process would be repeated over and over again until satisfactory results are achieved.

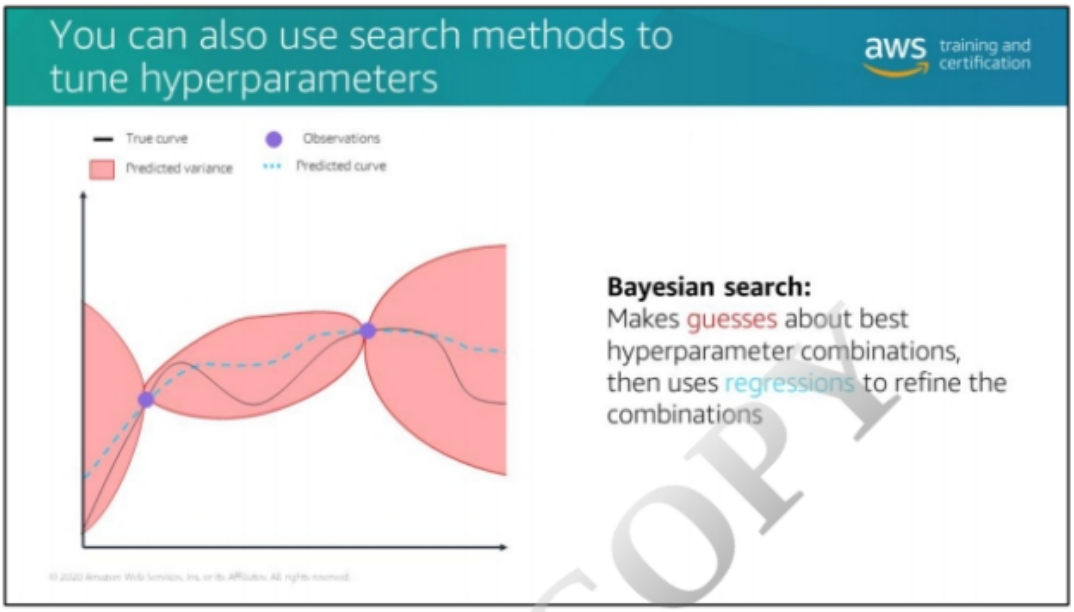
Needless to say, this is not always the most thorough and efficient way of tuning your hyperparameters. As a result, several other methods for hyperparameter tuning have been developed.



Grid search is one of those methods. With grid search, you set up a grid made up of hyperparameters and their different values. For each possible combination, a model is trained and a score is produced on the validation data. With this approach, every single combination of the given possible hyperparameter values is tried. This approach, while thorough, can be very inefficient.



Random search is similar to grid search, but instead of training and scoring on each possible hyperparameter combination, random combinations are selected. You can set the number of search iterations based on time and resource constraints.



Bayesian search treats hyperparameter tuning like a regression problem. Given a set of input features (the hyperparameters), hyperparameter tuning optimizes a model for the metric that you choose. To solve a regression problem, hyperparameter tuning makes guesses about which hyperparameter combinations are likely to get the best results, and runs training jobs to test these values. After testing the first set of hyperparameter values, hyperparameter tuning uses regression to choose the next set of hyperparameter values to test. In this example, the Bayesian search runs initial training jobs to create the a predicted curve using objective function.

When choosing the best hyperparameters for the next training job, hyperparameter tuning considers everything that it knows about this problem so far. Sometimes it chooses a combination of hyperparameter values close to the combination that resulted in the best previous training job to incrementally improve performance. This allows hyperparameter tuning to exploit the best known results. Other times, it chooses a set of hyperparameter values far removed from those it has tried. This allows it to explore the range of hyperparameter values to try to find new areas that are not well understood. The explore/exploit trade-off is common in many machine learning problems.

